On the Provision of Safety Assurance via Safety Kernels for Modern Weapon Systems

J. Bret Michael[1], Anil Nerode[2], and Duminda Wijesekera[3]
[1]Naval Postgraduate School, Monterey, Calif., *bmichael@nps.edu*
[2]George Mason University, Fairfax, Va., *dwijesek@gmu.edu*
[3]Cornell University, Ithaca, N.Y., *anil@math.cornell.edu*

Abstract*

In this paper we discuss some of the challenges and approaches for providing safety assurance for modern weapon systems via software-based safety kernels. We argue that software-centric approaches for designing and verifying safety kernels are flawed. We claim that the design and verification of safety kernels for complex event-driven real-time systems is a matter of physics and dynamical system analysis of system design. We describe an approach for rapidly prototyping safety kernels (and plants and controllers) using an agent-based safety-kernel architecture. The approach utilizes multiagent modeling and hybrid automata.

***Keywords**: System safety, safety kernel, software, hybrid automata, verification*

1. Introduction

A *safety kernel* is a component specifically designed to reduce the probability of occurrence of mishaps by performing one or more of the following types of fail-safe functions in a system: detecting, tolerating, and isolating faults. The design of a safety kernel should be based on the following information obtained from hazard analyses of the target system:

- Hazard causal factors, along with frequency and severity of hazards and mishaps
- Complexity of the system protected by the safety kernel
- Number of safety-related functions in the system

Based on the foregoing information, decisions need to be made about the following:

- How much control is to be exercised by the safety kernel over safety-related functions?
- What type of safety-kernels architecture should be employed?
- Which parts of the safety-kernel functions should be allocated to software, hardware, and humans?

The safety kernel needs both sufficient control and functionality to return the system that has entered an unsafe state to a safe (or less risky) state.

So why in the past did so few safety-critical systems have safety kernels? Many factors played a role in stymieing the introduction of safety kernels into safety-critical systems. For example, consider design experience. There was no concept of a safety kernel in the analog version of legacy systems that have since been reengineered to incorporate digital technology. Another factor is engineering judgment. Up until recently there was little experience upon which to judge the prudence of using safety kernels rather than the tried-and-true safety-engineering techniques—the use of a safety kernel was viewed as posing an untenable level of risk not only by the developer and operator of a system, but also by safety certification and accreditation boards.

2. Safety Kernels for Weapon Systems

The first weapon within the U.S. Navy's arsenal to use a safety kernel was the fire-by-wire Rolling Airframe Missile (RAM) Guided Missile Weapon System (GMWS). RAM is quick-reaction, fire-and-forget missile designed to destroy anti-ship missiles.

The GMWS is an example of a complex event-driven real-time system. The system performs real-time processing of sensor data in order to detect, track, and target threat objects. GMWS and the RAM itself are both safety-critical because they control the release of energy—energy that can cause death or injury to humans, property damage, or damage to the environment. For example, both inadvertent launches and premature detonation of a RAM in close proximity of the mother ship are hazards which need to be controlled.

Let's take a look at the Launcher Control/No-Point No-Fire (LC/NPNF) subsystem of the GMWS. The LC/NPNF system monitors the potential movement of the missile launcher into a NPNF zone. The launcher is mounted to the deck of a ship. The NPNF zone consists of the deck, the ships superstructure (e.g., bridge, antennas), and other areas

* The views and conclusions contained in this presentation are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

on deck such as where sailors may be located. A safety incident occurs if the launcher points in a NPNF zone. If the launcher points or fires in a NPNF zone and fires, a mishap is almost certain to occur.

Prior to RAM GMWS, the LC/NPNF subsystem was controlled via electromechanical mechanisms, including metal cams, switches, and roller contacts. However, the U.S. Navy found that its reliance on electromechanical means for enforcing NPNF safety policy was not a cost-effective approach if modifications were made to a ship's superstructure or the weapon needed to be installed on another class of ship (e.g., cruiser, frigate). In other words, any changes to the superstructure or movement of the weapons system to either a different mounting position or another platform would require the reengineering of the electromechanical system. This was one of the reasons the U.S. Navy assumed the risk of introducing the use of software-controlled dynamic NPNF zones.

The software-controlled NPNF function of the LC/NPNF subsystem is an example of a safety kernel. The NPNF safety kernel detects the potential movement of the launcher into a NPNF zone. If the NPNF safety kernel determines that the Launch Control System (LCS) will move the weapon into the NPNF zone, the kernel takes control of the LCS and executes an orderly shutdown of the GMWS. However, the NPNF processor does not provide the fidelity or control necessary to train or elevate the launcher to prosecute an engagement. The NPNF processor performs the following two tasks: (i) stops the launcher movement and (ii) interrupts concurrently the Launcher Control Processor and the firing circuit to the missile (in order to preclude arming and firing of the RAM).

The use of a software-controlled NPNF safety kernel for the LC/NPNF subsystem is a double-edged sword: it increased the complexity of the system (i.e., in terms of state-space) and approximately doubled the size of control software of the GMWS, but on the other hand it reduced both the overall level of risk of putting the system into operation and the need to make costly changes to the GMWS hardware or the ship's superstructure.

3. EXAMPLES OF SAFETY-KERNEL ARCHITECTURES

At one end of the spectrum of safety-kernel architectures is the watchdog safety kernel. This kernel has limited functionality: its primary function is to detect failures and either reset the processor or throw an exception to terminate a process.

At the other end of the spectrum are dual and multiple redundant architectures, which may consist of homogeneous or heterogeneous safety kernels. Dual redundant safety kernel architectures only provide for fault detection. In contrast, multiple redundant safety kernel architectures provide for fault detection, fault tolerance, and fault isolation.

Examples of safety kernel architectures that lie within the middle of the spectrum are safety executives and monitor-actuator patterns. The former initiates fail-safe processing: monitors the state of a system and ensures that the software cannot enter a potentially unsafe state, in addition to coordinating recovery from faults. The latter returns a system to a known less-risky state and resumes processing via monitoring the actuation functions of another process and the state of the actuators. The RAM GMWS LC/NPNF is an example of the application of the monitor-actuator pattern.

4. NEED FOR A NEW APPROACHES TO PROVIDING SAFETY ASSURANCE

Safety design requirements are typically easier to implement in the weapons system or weapons-related system than in an external system. The way in which safety design requirements are implemented in U.S. Navy weapons systems is relatively homogeneous: known safety attributes and characteristics of these systems are already relatively well known to the system safety programs.

So why consider the use of safety kernels? Assessing the level of control to be exercised over the weapons or weapons-related system becomes increasing challenging as the level of system integration and complexity increases. Such assessments are especially problematic to perform for system-of-systems. A *system-of-systems* is an amalgamation of legacy systems and developing systems that provide an enhanced capability greater than that of any of the individual systems within the system-of-systems. Systems-of-systems are a great departure from standalone systems. There is uncertainty and risk associated with assumptions and unknowns regarding the interfaces between the component systems. There is also uncertainty and risk associated with system interoperability issues.

Let's take a look at a real-world system-of-systems—the Ballistic Missile Defense System (BMDS). Like the RAM GMWS, the battle manager element of the BMDS is a real-time, event-driven complex system. However, it is also asynchronous and distributed. The battle manager must interface to a large number of heterogeneous legacy, organic, and even foreign systems, some of which may not have undergone sufficient safety assessment. In addition, the configuration of the system needs to adapt via plug-and-play components to changes in the environment; that is, the system needs to be readily reconfigurable during operation. Thus, traditional approaches for providing safety assurance of BMDS will not be cost effective and do not lend themselves well to verification.

5. WHAT IS THE WAY FORWARD?

System safety relies on predictability. There is a need to know what the system must guard against (i.e., hazards). How does one handle unanticipated hazards? Adaptive systems can have lots of configurations; it is hard to characterize these configurations because each instance of a component has a different view of the system.

Moreover, the set of things in the environment is neither closed nor stable. However, it might be possible to create a sufficiently large closed world so that one can deal with all of the system hazards. Even so, there will still be a challenge to validate an upper bound on the probability of a system failure leading to a given hazard.

The lack of predictability is at odds with the view of a system for which safety, reliability, and other forms of dependability engineering (i.e., dependability encompasses all of the "ilities") typically rely.

One of the promising new approaches to providing safety assurance for weapon systems is to think in terms of dependability disciplines.

5.1 Challenges

In order to explore a dependability-disciplines view of providing safety assurance for weapon systems, one must treat a system or system-of-systems in terms of integration properties in at least two respects: (i) to identify the emergent requirements of the weapon system from the collaborations (i.e., determine value-added, rather than what must not happen) and (ii) to certify that the legacy, organic, and foreign systems meet constraints of the well-defined "plug-in slots" (i.e., system interfaces) of the plug-and-play reconfigurable system.

We can construct a wish list for weapon systems, to include, for example, the following desires:

- Coordinated battle management at system-of-systems level vice system level
- Predictable behavior of system-of-systems
- Integrated systems vice interconnected systems, bringing together legacy systems, new system developments, and nondevelopmental (e.g., commercial- and government-off-the-shelf) items
- Minimal effort for modifications to system-of-systems
- System architectures that outlive their components

5.2 Achieving Dependability[2]

We advocate a departure from business as usual in the engineering of weapon systems, especially those that a part of systems-of-systems, by requiring the following:

- Spartan and Draconian designs of the system
- Distinguishing up front which system requirements are stable from those that are expected to change
- Institutionalizing the invariant part of the principles of operation of the system
- Taking a positive approach to handling emergent properties of systems, thinking in terms of integration
- Defining emergent requirements and ensure realization

A *Spartan safety kernel* provides for liveness properties with service guarantees. The Spartan safety kernel only provides services needed to achieve critical requirements in a timely manner.

A *Draconian global structure* provides for safety with non-interference guarantees. The structure affords for the visible dependencies to be much less than potential dependencies, with fault containment at boundaries and no invisible interactions.

The *safety executive* resides within the Spartan safety kernel. The safety executive monitors in a cyclic fashion the high-level functionality (i.e., execution of high-level processes) of the system or system-of-systems to ensure that the processes follow the desired sequence of execution. For example, in the case of ballistic missile defense, the safety executive of the safety kernel associated with one of the replicated battle managers would monitor whether the battle manager's processes for prosecuting a missile engagement execute in the required manner.

5.3 Battle Manager Development

In addition to the Spartan safety kernel, we foresee the need for the weapon system to also have one or more *battle management kernels*—kernels that contain only the basic functions of battle management. Derived from the kill chain, these basic battle-management functions will manage the use of the system's computing resources to ensure that all time-critical, battle-management events are processed as efficiently as possible. All other weapon-system functionality is to be placed in components that interface with the battle management kernel via the aforementioned well-defined plug-in slots. The Spartan safety kernels monitor the behavior of the battle management kernels, taking action as needed to enforce safety policy, while the battle management kernels are responsible for monitoring the behavior of the weapon system or system-of-systems of weapons, taking action as needed to enforce the rules of engagement and other policy and doctrine related to prosecuting an engagement.

[2] The concept of a Spartan safety kernels and Draconian design of safety-critical systems emerged during discussions between Valdis Berzins and Bret Michael. Bret Michael presented these concepts in the context of missile defense at the Defence and Aerospace Research Partnership in High Integrity Real Time Systems (DARP HIRTS) Workshop, Manchester, England, on May 5, 2004.

6. Design and Verification of Safety Kernels

Existing software-centric approaches for designing and verifying safety kernels are flawed: these approaches rely on models of faults which occur at discrete times and can be identified. This has little to do with systems that are governed by a continuous sequence of messages which alter the controlled behavior of devices (e.g., networks of weapons and sensors). In essence, today's popular approaches are based on verifying that a pure software system obeys its software specification and does not send out ineligible signals.

What is needed is a verification approach that provides the dependability engineer with the ability to determine what sequences of control actions would cause a catastrophe. No one control action in the sequence has any meaning, but the whole sequence may send the system spiraling out of control (i.e., positive feedback, not control).

However, we do not believe that the design and verification of safety kernels is software problem. We claim that the design of safety kernels for complex event-driven real-time systems is a matter of physics and dynamical system analysis of system design.

For verification purposes, one needs to provide simulation or mathematical evidence for complex systems that the real systems of physical devices are similar enough to their finite automaton approximations so that the finite automata controls would control the system in the real world.

Hybrid automata are needed for prototyping and verifying safety kernels. The automata can be used to represent complex real-time systems as distributed systems of interacting physical (e.g., sensors and launch systems) and rule-based (e.g., decision makers and threat evaluators) agents. Physical devices are modeled as Buchi finite state automata on infinite strings; Buchi finite state automata differ from finite automata in that they operate on infinite words and have a different acceptance condition. Agent networks of Buchi automata constitute a special case of Rabin automata (have strong-fairness acceptance) which is computationally feasible and represent many significant aspects of multiagent systems.

Doctrine, policy (including safety policy) and organizational structures are treated as rule-based constraints on system behavior.

Every agent is modeled as Datalog program, which can then be used to run simulations with the aim of determining the effects of inserting, modifying, or deleting physical and rule-based agents. This approach provides a means for rapid prototyping of safety kernels (and plants and controllers) using an agent-based safety-kernel architecture.

We believe that Datalog is a good choice of modeling languages because it is expressive enough to represent the agents, but yet compiles decently to real-time deterministic Buchi automata.

The next step we intend to take is to develop a professional-grade simulation test bed, with the aim of providing a means for verifying the design of safety kernels (and plants and controllers).